PCT/GB 96/02977

Europäisches Patentamt | European Patent Office | Office européen des brevets

09/025581

REC'D 30 JAN 1997
WIPO PCT

Bescheinigung | Certificate | Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. | Patent application No. | Demande de brevet n°

95308682.4

PRIORITY DOCUMENT

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

[illegible]

SCHMITT C.H

DEN HAAG, DEN
THE HAGUE, 09/01/97
LA HAYE, LE

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 95308682.4
Demande n°:

Anmeldetag:
Date of filing: 01/12/95
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
BRITISH TELECOMMUNICATIONS public limited company
London EC1A 7AJ
UNITED KINGDOM

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
Data access

In Anspruch genommene Prioriät(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
G06F17/30

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/DE/DK/ES/[illegible]/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

DATA ACCESS

The present invention relates in general to data access techniques in computer systems.

Generally, the least expensive memory in a computer system is also the slowest and, unfortunately, system performance suffers when a fast device must wait for a memory system to access data. The device may be an I/O device or a CPU and the memory system may be main memory, for example RAM, or external storage, for example a hard disk drive. By way of example, it is not uncommon for a CPU to operate at speeds in excess of ten times that of even very expensive, fast memory. Thus, computer operations which require access to stored data can suffer a memory-access bottle-neck. Computer system designers, therefore, have to consider a trade-off between system performance and memory cost.

The use of data caches in computer systems is widespread. A cache provides one way of increasing computer system performance without undue extra cost. Cache memories are, typically, high-speed buffers for holding copies of the most recently or most commonly accessed data in main memory or in an external data storage device. By adding a cache memory between a fast device and a slower memory system, a designer can provide an apparently fast memory system.

The use of caches improves system performance in several ways. Most obviously, a cache can speed up a system by increasing the access speed to the most commonly used data. Thus, the overall average data access speed of the system is increased. Also, if cached data only is accessed whenever possible, this cuts down on the data transfer overhead between the accessing device and the slower memory system. This second aspect is particularly significant in multiple-user systems, for example client/server environments, where the users may wish to access central data stored in a central database on one or more network file-servers: a reduced data transfer overhead conserves precious network bandwidth.

In any cache system, a major consideration is cache coherency (also known as cache consistency). That is, is the data accessed in the cache consistent with, or rather the same as, the respective data in the slower memory system?

In client/server systems, there is considerable opportunity to maintain with the client a large cache of recently read data. This cache may be used during an active transaction to avoid re-reading data from the central database. However, if data is cached on a client and the client needs to re-access that data, the assumption must be made that the data on the cache is 'stale', because there is no way of knowing differently. This means that a read access is needed to read the current state of the data in the central database.

Static data, that is data which rarely changes, are prime candidates for holding in a client cache database. The data can always be assumed to be current. However, a task must be set in place to ensure that any changes to the static data are propagated to all client caches. This is typically done on an overnight batch run basis. In these cases, there is no need for a real-time process to maintain consistency.

Highly dynamic data is extremely difficult to maintain in a consistent state. The basic reason is that if the data changes very often, the network and processor impact, in a client/server environment, of up-dating many other caches can be considerable. In some cases, the cost of maintaining the caches might exceed the cost of accessing the central database for the current data.

In between static and highly dynamic data is a type of data which is not static, but which does not change frequently. Typically, in this case, data might be held on the central database, but then might be cached on a client, for example a travel agent, during, for example, the period of an aircraft flight enquiry and the booking of a seat on a flight as a result of the enquiry. Then, there would be a high degree of certainty that the data remains consistent during the operation. However, there would never be total certainty because, coincidentally, another client, or travel agent, might book the only remaining seats on the respective flight after the first client began its operation but before the operation was completed.

The simplest way to prevent data inconsistency is to 'lock' any data on the central database which is being accessed by a client, thus making that data inaccessible to other clients. This is called 'pessimistic locking'. Typically, a lock table is generated on a file-server which holds the identities of locked data. Such a system requires that all access requests made by clients invoke a lock table search on the file-server before data access is allowed, or denied.

Obviously, however, in an environment where the same data might need to be accessed or up-dated by several clients, for example for flight-booking purposes, pessimistic locking represents an unworkable solution with intolerable locking overheads.

Another method of dealing with the possible inconsistency between cached and central data is discussed in the book "Transaction Processing - Concepts and Techniques" by Gray J and Reuter A, published by Morgan Kaufmann 1993, on pages 434 to 435. The method involves 'optimistic locking' using time stamps. Optimistic locking allows clients connected to a server to use cached data for reading purposes, but as soon as a transaction is initiated, for example a flight seat booking needs to be made, the cached data is compared with the central data to ensure data integrity and the central data is locked for the period of the actual transaction. This prevents another client from changing the central database version of the data during the transaction. If it is found that the data in the cache database is different from that in the central database, the cache database is updated and then the client is notified and left to sort out any problems this inconsistency might cause.

The advantage of optimistic locking is that the central data is only locked for a very short period of time, for maybe less than one second to carry out the actual transaction. In contrast, using pessimistic locking, which requires the data accessed to be locked for the whole period of an operation, would result in the data being locked for the whole period of an enquiry and a transaction, which might take many minutes.

Therefore, although optimistic locking may require extra processing by both a client and a server, it can be seen that the technique is far more suitable for dealing with relatively dynamic data which needs to be accessible to multiple clients.

Sometimes, when implementing optimistic locking, time stamping is used to mark data rows in a cache database with the time the data was last updated, and to mark the corresponding data rows in the main database with the respective last up-date time. In this way, if the time-stamps for a particular row of data held in both the cache and main databases are the same, the system knows that the cached data is current and there is no need to send the data in question across the network from the client to the server to facilitate comparison. Thus, network bandwidth is conserved whenever a cached copy is found to be current.

In accordance with a first aspect, there is provided a method of accessing data in a computer system, the method comprising the steps of a issuing a read request for data, the request including a first key representative of the state of an existing copy of said data stored in a first data area, searching a second data area for an index reference to the location of said data in a third data area, said index reference including a second key representative of the state of the respective data and, if said first and second keys are equivalent, indicating that the existing copy of said data is valid.

According to a second aspect of the present invention, in a computer system including a first memory means and a second memory means, the first memory means comprising a memory area for storing data and the second memory means comprising a first memory area for storing data and a second memory area for storing indices, each index pointing to the location of a piece of data in the first memory area, wherein data held in the first memory means comprises a sub-set of the data stored in the second memory means, there is provided a method of accessing stored data, the method including the steps of:

in response to a request to read a piece of data from memory, searching the first memory means for said piece of data and, if present, retrieving a first key stored in association with said piece of data;

searching the second memory area of the second memory means for an index to said piece of data stored in the first memory area and retrieving a second key stored in association with said index; and

if the first and second keys are the same, providing an indication that the piece of data from the first memory means is valid.

According to a third aspect of the present invention, in a computer system including a first memory means and a second memory means, the first memory means comprising a memory area for storing data and the second memory means comprising a first memory area for storing data and a second memory area for storing indices, each index pointing to data in the first memory area, wherein data held in the first memory means comprises a sub-set of the data stored in the second memory means, there is provided a method of accessing stored data, the method including the steps of:

in response to a request to write a piece of data to memory, writing said data to the first memory means;

writing said piece of data to the first memory area and writing a respective index to the second memory area of the second memory means, and providing a first key to be stored in association with the index; and

writing a second key to be stored in association with said piece of data in the first memory means.

According to a fourth aspect, the present invention provides a computer system including first memory means and second memory means, the first memory means being arranged to store therein a subset of the contents of the second memory means, the second memory means comprising a first memory area for storing data and a second memory area for storing indices, each index pointing to a specific piece of data in the first memory area, characterised in that at least some of the data in the first memory means includes a first key indicative of when the respective data was last updated and at least some of said indices include a second key indicative of when the respective data in the first memory area was last updated.

According to a fifth aspect, the invention provides a computer memory system including:

a first data area for storing data records, at least some of which include a first key representative of the state of the respective data record;

a second data area for storing index references, at least some of which include a second key, each of said index references indicating the location of a respective data record stored in a third memory area, each said second key being representative of the state of the respective data record in the third memory area;

means for receiving a request for a data record, said request including a first key associated with a data record stored in the first memory area;

means to compare said first key with a second key associated with the index reference of said requested data record; and

means operable, if said first and second keys are equivalent, to indicate that the data record in the first data area is valid.

A key preferably comprises a time-stamp which relates to the last time its respective data was up-dated. Alternatively, however, a key might be some other indicator of when the respective data was last up-dated. For example, a key might comprise an incremental counter which is incremented whenever the respective data is amended.

In preferred embodiments the invention is implemented in a multiple user computing environment, for example a client/server environment. Preferably, the first memory means is a memory associated with a client and the second memory means is a memory associated with a server. In either case, the memory might comprise main memory, for example RAM, an external data storage device, for example a hard disk, or a combination of the two. The computing environment might be, for example, a network database system in which multiple clients have access across a network to a database stored on a central file-server. A suitable network might be a local area network, but could easily be a metropolitan area network, a wide area network, or even an international or global communications network.

Preferably, the first memory means holds a cache database associated with a client. The database would, typically, be held on an external storage device, for example a hard disk drive.

In the case of a client/server environment, the second memory means preferably holds a central database to which clients have access. The central database typically comprises an index area and a data area, the index area being stored in the first memory area and the data area being stored in the second memory area.

Advantageously, having keys associated with entries in the index area in the first memory area obviates the need to read data from the data area in the second memory area in the event the keys are found to match or are equivalent. Hitherto, use of keys such as time stamps has been limited to associating a key with the data in the data area only, and not the index area. Therefore, regardless of whether the keys were the same, the data would have needed to be read from the data area.

In some embodiments, the second memory means comprises main data storage in an external storage device and buffer data storage in main memory.

Preferably, the first memory area resides substantially in the external storage device.

In preferred embodiments, the invention is implemented in the memory system of a suitable computer system. Preferably, the first memory means comprises a memory cache and the second memory means comprises main memory. In some cases, main memory might also comprise virtual memory, for example paged memory on an external storage device such as a hard disk drive.

Preferably, the memory cache is comprised by relatively fast access memory such as, for example, static RAM. The main memory is preferably comprised by relatively slow access memory such as, for example, dynamic RAM.

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings, of which:

Figure 1 is a block diagram of an exemplary, two-tier client/server computing environment in which the present invention can be implemented;

Figure 2 is a diagram which represents a simple memory arrangement for a database system;

Figures 3a to 3c are diagrammatic representations of the row and index structures of a database for two prior art systems and the present invention respectively;

Figure 4 is a flow chart representing a typical database transaction;

Figure 5 is a flow chart which represents a typical database read process;

Figure 6 is a flow chart which represents a modified database read process;

Figure 7 is a flow chart which represents a write process;

Figure 8 is a diagram representing a three-tier architecture on which the present invention can be implemented; and

Figure 9 is an alternative arrangement to Figure 8.

Figure 1 illustrates an exemplary, two-tier client/server computer environment in which the present invention may be implemented.

A file-server 100, for example a computing platform running the UNIX operating system, runs network and database management system (DBMS) software suitable for providing remote database access to a plurality of clients 130 over a network 140. In this description, unless otherwise stated, the term "client" will be used to describe both the physical computer 130 connected to the network 140 and an operator of the computer. In a local area network environment, a suitable network might be a co-axial Ethernet network running TCP/IP software, and suitable DBMS software might be Oracle version 7. DBMS software allows creation and control of complex database systems and access to the data therein. For further information on such systems, the reader is referred to the numerous texts and reference manuals on database systems which are widely available.

The file-server 100 houses standard computer components such as a processor 102, main memory 104 and input/output (I/O) hardware 106, all connected by suitable data and address buses 108. The file-server supports, typically, a large capacity (for example 10 Gbytes) external storage device such as a hard disk 120, on which substantially all the information in a central database is stored.

The main memory 104 of the file-server is split into areas for standard program and data storage. However, for the example of running Oracle database software, the main memory includes a large area of memory known as a system global area (SGA) 105, the purpose of which will be explained below. In this description, the terms "file-server", "database" and "DBMS" may be used interchangeably and all refer, in general, to a central data storage system to which clients have data access.

The clients 130 are standard computer systems, for example IBM Personal computers, each including a standard hard disk drive 135 having around 100 Mbytes of storage capacity. The clients 130 run software, for example in a Microsoft Windows environment running under MS DOS, suitable for accessing (reading and writing) data stored in a database using structured query language (SQL) queries which are understood and processed by the DBMS software. The hard disk drives 135 have a storage area for a cache database 136, into which data from the central database 126 can be copied and accessed as required.

As illustrated in Figure 2, typically, the central database 126 comprises two main physical areas. The first area is the data table 204, where the database data is stored. A data table 204 typically comprises a group of related data. For example, a table 204 may hold the information about all scheduled aircraft passenger flights around Europe in the coming year. A table is split up into pages

215, or blocks, comprising rows of data, where, in this example, one row 210 would represent one particular flight. The data about each flight would be held in fields 305 within each row 210 as illustrated in Figure 3a. Example fields 305 for a particular flight might include:

destination	3 bytes
origin	3 bytes
in week commencing	2 bytes
date	8 bytes
time	5 bytes
seat availability	3 bytes
flight number	8 bytes
price per seat	8 bytes
total	120 bytes

The second area in the database is the index table 206 comprising index pages 220. The index pages 220 contain indices 225 for each row 210 in the data table 204. An index 225 typically comprises index fields, one of which references a specific row of data in the data table 215.

As illustrated in Figure 3a, an index 225 includes a first index field 310, for example containing a destination, a second index field 311, for example containing an in week commencing value, and a third index field 312 which is a pointer to where the corresponding data row 210 is held in the data table 204. In this example, therefore, flights can be searched by destination and in a particular week of the year. An example search query might specify all flights to Paris in the 25th week of the current year. The DBMS, in response, would search the index for all flights to Paris in the week commencing 10 December 1995 and return the results to the requesting client. Then, the DBMS would access the database rows determined by the index search. Obviously, other search criteria, for example by specific date, would require extra index fields included in the index table configuration.

The cache database 136 comprises a storage area 230 for cached data. The cached data typically comprises data rows 235 copied from the central database 126. The rows 235 themselves are typically exact copies of the rows 210 in the data table 204, although this need not be the case. Thus, the row configuration illustrated in Figure 3a is common to both the cache and the central database data rows.

The cache database 136 may or may not have a corresponding index table, depending on the size and acceptable access speed of the cache database. For the present purposes, it will be assumed that the cache database has no index table.

The SGA 105 memory area in the main memory 104 on the file-server 100 is used to speed up central database access for clients 130. DBMS other than Oracle implement similar schemes. Whenever a client requests data access, the file-server 100 looks initially in the SGA 105 for a copy of the data row or rows. If the data is not present, the file-server accesses the central database 126 and copies the data to the SGA 105. When a row needs to be copied from the central database 126 to the SGA 105, in fact, the whole page containing that row is copied to the SGA 105. From there, any access of the required row or rows is made in the SGA 105. In this way, over time, the most commonly accessed and/or dynamic data will reside on the SGA 105 and, since access to main memory 104 can be in excess of an order of magnitude faster than hard disk 120 access, the overall average access time for the database is reduced. Obviously, main memory, at present, is limited in size far more than external storage systems. Therefore, the SGA 105 can only contain the most recently-used data. Also, since main memory 104 tends to comprise RAM, which is volatile, the database needs to be updated regularly with any changes to data held in the SGA 105. In effect, the SGA 105 forms part of the database and acts in a similar way to a 'write back' cache in that data is periodically written back to the main database area 122 on the disk drive 120.

In practice, an SGA will hold any data which is accessed from the central database 126, including data pages and index pages. For simplicity of description, it will be assumed in the following description, unless otherwise stated, that the SGA 105 is always searched before the external storage device is searched.

As already stated, typically, a cache database 136 has a cache table 230 arranged to hold data rows 235 which are exact copies of data rows 210 from the central database 126.

In some embodiments, however, it is possible that the cache database is arranged to receive whole pages from the SGA 105 rather than single rows. This would be advantageous, for example, if data access tended to be to clustered data. However, it will be assumed in the following description that only rows are cached.

Whenever a client 130 requires data, the cache database 136 is searched through first. If the data required is present in the cache database then it is retrieved therefrom and used, as required. If the data is not in the cache database 136, then the central database 126 is accessed (via the network 140, the file-server 100 and the index table 220) and the current data (row or rows) is copied to the cache database 136, from where it can be retrieved and used, as required.

Problems with this arrangement arise if it is possible that more than one client 130 can access and alter data held on the central database 126. Then, any cached copies of that data held by a client 130 on a client cache database 136 would become invalid, or stale. This can be overcome using the locking techniques described above.

In the following description, data access denial due to the existence of a lock, and the corresponding checks, are not considered. It should be remembered, however, that in practice a lock might prevent a desired data access and that the DBMS usually deals with such a situation in an appropriate manner, for example by freezing the operation of the denied client until the lock is removed.

The two main operations carried out by a client in a database environment are read data and up-date data: create or write data will for the present purposes be treated in the same way as up-date data. A combination of read and write typically forms the basis for client/server database transactions. An example of a transaction is given in the flow chart in Figure 4.

In Figure 4, a client 130 transmits a query, in step 400, to read data from the central database 126. In step 410, the file-server 100 receives the query. The file-server 100 provides the information, in step 420, to the client 130. The client then, in step 430, carries out whatever business logic may be required on the data so supplied. An example of simple business logic might be deciding which flight to book, from a number of available flights to Paris, on the basis of price. In step 440, a transaction is initiated by the client 130. The client 130 transmits data to the file-server 100. Data might include, for example, customer details of a person who has booked a seat on a flight to Paris and an amendment request to reduce seat availability on that flight accordingly. On receipt of the data, in step 450, the file-server 100 searches for and locks the data to be up-dated. Then, in step 460, the file-server 100 up-dates the flight data which is held in a row of a data table, if possible, and adds the new data to a customer data table. In step 470, the file-server 100 commits the new and up-dated data to record. The records are then unlocked, in step 480, to allow access by other clients to the data. Finally, in step 490, the file-server 100 transmits a status response to the client that the transaction has been successful.

For the purpose of comparison, Figures 5 and 7 illustrate in flow chart form respective read procedures in response to a suitable read query for a prior art system and a system according to the present invention. In both examples, only one data row is requested but it will be appreciated that queries for multiple rows are a possibility. Also, it is assumed that a cache copy of the data exists in the cache database 136 and that both cache copies of data and central copies of data include time stamps which reflect the last up-date time for the respective data.

The description below of the process in Figure 5 assumes that the data rows and indices have the structure shown in Figure 3b.

In Figure 5, in step 500, the client 130 transmits a query, across the network 140 to the file-server 100, to read data from the file-server. The query includes an identifier for the data row required and a respective time stamp. In step 505, the file-server 100 receives the query. The file-server 100, in step 510, accesses the SGA 105 in search of the index 225 for the data in question and retrieves the index if it is present. In step 520, if the index 225 is present in the SGA 105, then on the basis of the index value, the file-server 100 accesses the data table 204 of the central database 126 to retrieve the data page 215 containing the row 210 into the SGA 105, in a step 540. On the other hand, if the index 225 is not present in the SGA 105, then, in step 530, the file-server 100 accesses the index table 220 of the central database 126 to retrieve the index and then, in step 540, the file-server 100 accesses the respective data page 215 of the central database 126 to retrieve the data into the SGA 105. In step 550, the file-server 100 compares the time stamps of the cached data row with the central data row requested. If the time stamps are the same, the file-server 100, in step 570, sends a reply to the client 130 that the cached data is still valid, or, in step 560, if the time stamps are different, the file-server 100 transmits the entire data 210 to the client 130 to update the cache database 136. In step 580, the client receives the response and acts accordingly.

The description below of the process in Figure 6 assumes that the data rows and indices have the structure shown in Figure 3c.

In Figure 6, steps 600 to 620 are equivalent to steps 500 to 530 in Figure 5. In step 625, the file-server 100 compares the time stamp in the index with that of the query. In step 640, if the time stamps are the same, the file-server 100 transmits a cache still valid message to the client 130. On the other hand, if the time stamps are not the same, the file-server 100 has to access the central database 126, in step 630, to retrieve the current data row. The current data is then transmitted to the client 130 to update the cache database 136 in step 635. In step 645, the client 130 receives whatever data is returned by the file-server 100 and acts accordingly.

The advantages and disadvantages of the method and database arrangement of the invention can be appreciated by comparing the flow charts in Figures 5 and 6, and by considering the field lengths shown in Figures 3a to 3c.

Obviously, using the row configuration illustrated in Figure 3a has the disadvantage that a whole row of data would need to be passed from the client 130 to the file-server 100 to check for data consistency. However, using the configuration of 3b, there is only a requirement to transmit a reference indicating the required row, and time stamp. Thus, the transmission overhead between Figures 3a and 3b is cut by 107 bytes (that is the difference between a whole row of 120 bytes and a reference, comprising a destination field of 3 bytes, a week commencing field of 2 bytes and a time stamp field of 8 bytes ie. 120-13=102 bytes).

With reference to Figures 5 and 6 which use the data configurations of Figures 3b and 3c respectively, initially, the transmission overhead from the client

130 to the file-server is the same. That is, the steps 500 to 530 and steps 600 to 620 are equivalent. The first difference is between steps 540 and 625. In step 540, the file-server 100 reads the whole data page containing the required data row from the data table 204 to enable comparison of the data row 210 time stamp with the time stamp in the query. Therefore, in Figure 5, the file-server 100 must read the whole page from the central database 126, which is typically from the external storage device 120, to enable a comparison of the time stamps. This is a relatively time-consuming operation compared to accessing only an index which might be in the SGA 105 in main memory, as has already been discussed.

In contrast, in step 625, the file-server 100 compares the query time stamp with the index time stamp. The file-server 100 only needs to read the whole data page and row from the central database 126 in the event the time stamps are different. Consequently, external storage device 120 access is generally not required if the time stamps are the same. Therefore, in theory, the database access times and the processing overhead of the file-server 100 can be reduced and the response times for clients 130 can be increased.

A disadvantage of the proposed method is the increased size required for the index table. By considering the index arrangements of Figures 3b and 3c, it can be seen that there is approaching a 50% increase in index size. Therefore, theoretically, the index table size will increase by nearly 50% and the search time might increase proportionally. However, in practice, index tables are more complex and typically include many more index entries to allow more flexible query options. For example, a typical index might allow rows to be searched on date and flight number as well as destination and week commencing. Thus, the addition of a time stamp might not incur such an overhead. In general, it can be seen that the overhead of adding a time stamp to the index table depends on the size and complexity of the index table.

In some embodiments, it is only necessary to add time stamps to indices for data which is classed as dynamic, such as, for example, flight seat availability. Whereas, it would not be necessary to add time stamps to indices for static data such as, for example, flight number and departure date data. In this way, the time stamp overhead in the index tables may be reduced. Obviously, the SQL software and the DBMS would then need to deal with both possibilities.

While the example of using time stamps has been explained in detail, it will be appreciated that other methods for marking data rows to enable data consistency comparisons are possible. For example, an incremental field might be used instead of a time stamp. Each time a data row is amended, the count value in the field would be incremented. Thus a comparison between the increment value of a cache data row and the increment value of a central data row would highlight any changes to the central data row. Other comparable methods of identifying inconsistent data, for example using coding to produce unique codes representative of the specific state of a row, will be apparent to the skilled addressee on reading this description.

In some embodiments of the invention, the SGA 105 is large enough to contain, at least, the whole index table 206. This is highly desirable since all index accesses for reading or writing use only main memory 104 access and never have to access the index tables 220 in the external storage device 120. It is envisaged that this will become common practice in future when memory becomes cheaper and SGA sizes increase. Time stamping of indices which are all held in the SGA will therefore increase efficiency even more.

In any database system which allows data to be amended by a client, it is important for other clients to be able to access the updated data, if necessary, for carrying out a transaction. This would certainly be the case in the flight-booking scenario described above. In some scenarios, however, obtaining a current data value is not so important. This would be the case, for example, in a supermarket using point-of-sate (POS) terminals, or tills, which log all product sales to a database in the supermarket to facilitate automatic stock monitoring and ordering. In this case, each time a can of beans is sold the till logs the sale to the database, which acts as a cache database to a central database at the supermarket chain's headquarters. It would not be important for each till to have knowledge of the number of cans of beans remaining in the supermarket or indeed the number of cans of beans stocked by the supermarket chain. Therefore, up-to-date data consistency between the cache database and the headquarters central database would be unnecessary. All data could be written back (from the cache database to the central database) on a daily basis, which would be adequate to accommodate stock control for the supermarket chain.

For the present purposes, however, substantially only systems requiring consistent data, for example flight booking systems, will be considered in connection with up-dating data.

One solution for ensuring data integrity is to pass all changes back to the central database immediately they are made. The data up-dating process is explained below in conjunction with the flow chart in Figure 7.

In Figure 7, in step 700 the client 130 transmits a query to the file-server 100 to up-date certain data. The query includes a copy of the existing row, including the existing time stamp, and a definition of the details of the desired up-date. The file-server 100 receives the query in step 705. In step 710, the file-server 100 reads the respective data row from the central database 126, using substantially the procedure illustrated by Figure 6, whereby if the index time stamp is the same as the cached data row time stamp in the query, the cached data row is treated as the current row, otherwise the current row is read from the central database data table. The file-server 100 then locks the data row in the data table to disable access to it by other clients.

In step 715, the file-server 100 determines if the data row can validly be amended. Obviously, if the cached data row is deemed current then the up-date can validly be made. In the event the cached row is deemed not valid, and a data table read was required to obtain the current row, it is still possible that up-date can validly take place. One example of when an up-date would be possible, even if the cached data was not current, is if a client wants to book four seats on a flight, the cache database 136 reports that 100 seats are available but the central database 126 reports that only 90 seats are available: even though the cached data is stale, the transaction is still possible. If, however, a client wants to book ten seats on a flight, the cache database 136 reports that twelve are available but the central database 126 reports that only three seats remain, then the transaction cannot be completed. The DBMS software would in practice carry out a test to see whether a transaction is possible or not.

If an up-date is possible, in step 720, the row is amended by incorporating the amendment and by changing the time stamp. Then, in step 725, the time stamp in the corresponding index is amended to the same value. In step 730, the amended row and index are committed to the database 126 and SGA 105 respectively.

In step 735, the data row is unlocked to make it available for access by other clients. The file-server 100 then, in step 740, transmits a response to the client to acknowledge that the transaction has been completed. The response includes a time stamp value which matches the duly assigned time stamp of the amended row and index. The client, in step 745, receives the response, and in step 750 amends the row and updates the row time stamp with the new time stamp. In step 755, the client 130 writes the row to the cache database 136.

If the required amendment cannot be completed, in step 760, the file-server 100 unlocks the row and transmits, in step 765, a suitable response to the client 130. A copy of the current row is included in the response. In step 770, the client receives the response and, in step 775, updates the cache database 136 with the current row. In step 780, the client is left with the task of sorting out the discrepancy.

The write procedure described above ensures that all clients 130 are able to access current data if necessary. However, the procedure does incur an extra processing overhead on the file-server 100 above what would normally be required in a prior art system. The extra overhead is in step 725 in which the file-server 100 has to write to the index table 220 to update the index time stamp: normally, the index table 220 is only written to if the row data in the data table 215 is re-arranged for some reason, for example a new record is created or an old record is deleted.

It is envisaged, however, that as SGAs increase in size and more indices are stored in SGAs, this overhead will decrease in significance.

In some embodiments, rows of data may be arranged into a set which is stored in one or more pages of a database. This might be the case if a client commonly deals with several rows of data at the same time on a regular basis. Then, the client defines a single query which reads the set of rows, rather than requiring a separate query for each row each time. The file-server 100 has a matching set definition to support this facility. Each row in the set has its own time stamp. Thus, the overall time stamp of the set can be taken to be the latest time stamp of any row in the set. In this way, the database system need only compare set time stamps instead of the time stamps of each row to determine if the data is current. This raises the problem, however, that if another client deletes a row which is included in the set of another client, no latest time stamp is registered: the row is removed so there is nowhere for the time stamp to go. Therefore, the set time stamp remains the same even though the set has been amended (by deletion).

One solution to this problem is to use some form of cyclic redundancy check (CRC) algorithm to ensure that the CRC of some property of the rows in the set or the indices to the rows in the set is unchanged. The code could be incorporated into the time stamp field itself, or into a specific field. For example, a 32-bit CRC check would give a 1 in 4,294,967,296 chance of an undetected error.

So far in this description, the invention has been described in relation to a, so-called, two-tier computing environment in which a plurality of clients have access to a central database. The present invention is, however, not limited in any sense to use on such a system. For example, the present invention can be implemented on multi-tier systems. One such multi-tier system is shown in Figure 8.

In Figure 8, a back-end file-server 800 supports a database on a hard disk drive 805. The file-server 800 is equivalent in operation to the file-server 100 in Figure 1. The back-end file-server 800 is connected by a network 810 to mid-tier file-servers 815 which each have a hard disk drive 820 which includes a storage area for a cache database. The network 810 is an international network comprising private circuit connections supporting an ATM protocol. The mid-tier file-servers 815 operate in a similar fashion to the clients 130 in Figure 1.

To each mid-tier file-server 815 there is or are attached, via suitable networks 825, one or more clients 830. Suitable networks might comprise national networks such as, for example, a packet-switched data network. A client might be, for example, an IBM-compatible PC by which an operator can request information from the back-end file-server 800 database.

In this example, the clients 830 have access to the data in the back-end file-server 800 via the mid-tier file-servers 815. The clients 830 are unaware of the relationship between the mid-tier 815 and the back-end 800 file-servers: the relationship is said to be transparent to the clients.

An example of when such a system might be implemented is for an international company with local main offices in London (mid-tier server A) and Hong Kong (mid-tier server B) and a head office in Munich (mid-tier server C and back-end server 800). The clients 835 are based in respective regional offices which are connected to the local main offices via the national network 825.

In this example, the company's central database would be held on the back-end server 800 at the head-office. Thus, access by the clients 830 to static data would be direct from the mid-tier server 820 cache databases, onto which the necessary data is up-loaded regularly (for example, on a daily basis), or when required. Access to the more dynamic data would be from the back-end server 800, via the mid-tier servers 815, using the reading and writing processes described in relation to Figures 6 and 7 respectively. Thus, international direct access by the clients 830 to the back-end file-server 800 is restricted to that required to carry out transactions based critically on dynamic data. Again, a flight booking is a good example of such a transaction.

Another possible system on which the present invention may be implemented is that illustrated in Figure 9.

The arrangement illustrated in Figure 9 is a variation on Figure 8. The difference is the inclusion of a middleware system 908. The middleware system 908 is, in effect, an interface between the back-end file-server 900 database and the mid-tier file-servers 915. The middleware system 908 typically resides on a separate computing platform to the file-server 900. The middleware system 908 converts the format of data available from the back-end database, for example a legacy database system, to a format of data readable by the mid-tier file-servers 915 and their clients 930. Such an arrangement is also useful if the data required by, or the technology of the mid-tier or client systems change. Then, only the middleware system needs to be upgraded to accommodate the changes, rather than there being a need to up-grade the, typically, very expensive and complex, back-end database system.

Another variation on the system illustrated in Figure 8 is to incorporate a middleware system as a process on each mid-tier file-server. This has the advantage of removing the need for a dedicated middleware computing platform, but suffers with the disadvantage that more than one middleware process needs to be amended in the case of an up-grade.

A further variation on the systems illustrated in Figure 8 and 9 is that one or more of the clients 830 and 930 have there own cache databases on their hard disk drives 835 and 935. In this way, it would be possible for the clients to hold copies of data, typically static data, from the mid-tier or back-end databases. However, dynamic data could be stored in a client's cache database as long as cache coherency were maintained, for example by using the read and write procedures illustrated in Figures 6 and 7 respectively.

The skilled addressee will appreciate that the invention can be applied to almost any environment implementing cache databases, or equivalent. The decision to use the invention is purely one based on whether the invention provides an optimum system in terms of performance.

CLAIMS

1. A method of accessing data in a computer system, the method comprising the steps of:

a issuing a read request for data, the request including a first key representative of the state of an existing copy of said data stored in a first data area 230;

b searching a second data area 206 for an index reference to the location of said data in a third data area 204, said index reference including a second key representative of the state of the respective data; and

c if said first and second keys are equivalent, indicating that the existing copy of said data is valid

2. A method according to claim 1, including the step, if the first and second keys are not equivalent, of accessing the third data area 204, in dependence on the index reference, to retrieve a copy of said data.

3. In a computer system including a first memory means 136 and a second memory means 126, the first memory means 136 comprising a memory area 230 for storing data and the second memory means 126 comprising a first memory area 204 for storing data 210 and a second memory area 206 for storing indices 225, each index 225 pointing to the location of a piece of data 210 in the first memory area 204, wherein data held in the first memory means 136 comprises a sub-set of the data stored in the second memory means 126, there is provided a method of accessing stored data, the method including the steps of:

in response to a request to read a piece of data from memory, searching the first memory means 136 for said piece of data and, if present, retrieving a first key stored in association with said piece of data;

searching the second memory area 206 of the second memory means 126 for an index 225 to said piece of data 210 stored in the first memory area 204 and retrieving a second key stored in association with said index 225; and

if the first and second keys are the same, providing an indication that the piece of data from the first memory means is valid.

4. In a computer system including a first memory means 136 and a second memory means 126, the first memory means 136 comprising a memory area 230 for storing data and the second memory means 126 comprising a first memory area 204 for storing data 210 and a second memory area 206 for storing indices 225, each index 225 pointing to the location of a piece of data 210 in the first memory area 204, wherein data held in the first memory means 136 comprises a sub-set of the data stored in the second memory means 126, there is provided a method of accessing stored data, the method including the steps of:

in response to a request to write a piece of data to memory, writing said data to the first memory means 136;

writing said piece of data to the first memory area 204 and writing a respective index 225 to the second memory area 206 of the second memory means 126, and providing a first key to be stored in association with the index 225; and

writing a second key to be stored in association with said piece of data in the first memory means 230.

5. A method according to any one of claims 1 to 4, wherein the first and second keys are time-stamps representative of when the respective, associated data was last written to the respective memories.

6. A computer memory system including:

a first data area 230 for storing data records 235, at least some of which include a first key representative of the state of the respective data record;

a second data area 206 for storing index references 225, at least some of which include a second key, each of said index references indicating the location of a respective data record 210 stored in a third memory area 204, each said second key being representative of the state of the respective data record in the third memory area 204;

means for receiving a request 130 for a data record, said request including a first key associated with a data record 235 stored in the first memory area 230;

means to compare 102 said first key with a second key associated with the index reference 225 of said requested data record; and

means operable, if said first and second keys are equivalent, to indicate 102 that the data record in the first data area is valid.

7. A computer system including first memory means 136 and second memory means 126, the first memory means 136 being arranged to store therein a subset of the contents of the second memory means 126, the second memory means 126 comprising a first memory area 204 for storing data and a second memory area 206 for storing indices 225, each index 225 pointing to a specific piece of data 210 in the first memory area 204, characterised in that at least some of the data in the first memory means 235 includes a first key indicative of when the respective data was last updated and at least some of said indices 225 include a second key indicative of when the respective data in the first memory area 204 was last updated.

8. A computer system according to claim 7, wherein the first memory means 136 is a cache database and the second memory means 126 is a central database.

9. A computer system according to claim 8, wherein the cache database 136 is associated with a client 130 in a client/server environment and the central database 126 is associated with a server 100, wherein the client 130 and the server 100 are connected via a suitable communications network 140.

10 A computer system according to any one of claims 7 to 9, wherein the second memory area 206 resides substantially in main memory 104.

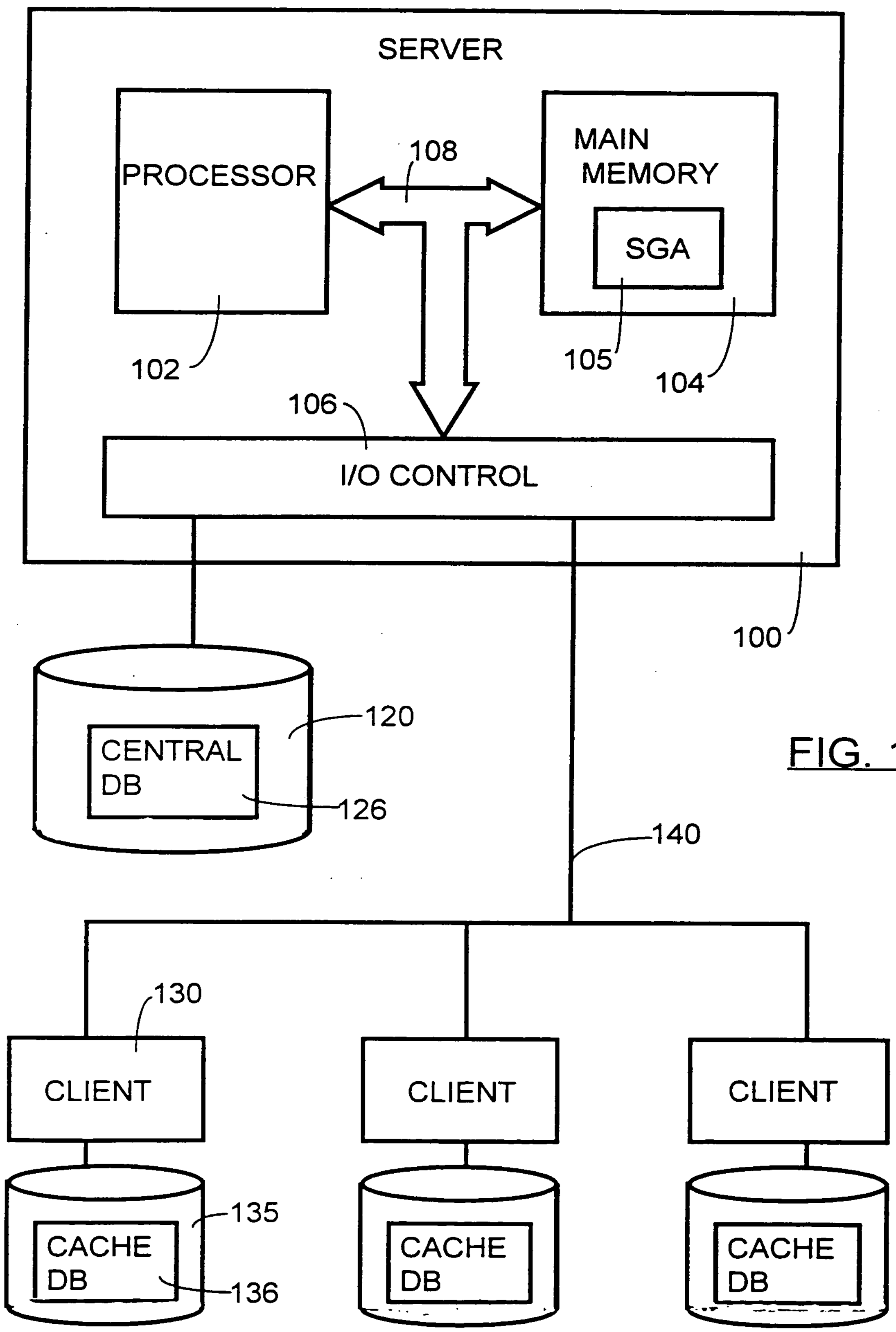

SERVER
PROCESSOR
108
MAIN MEMORY
SGA
102
105
104
106
I/O CONTROL
100
CENTRAL DB
120
126
140
130
CLIENT
CLIENT
CLIENT
135
CACHE DB
136
CACHE DB
CACHE DB

FIG. 1

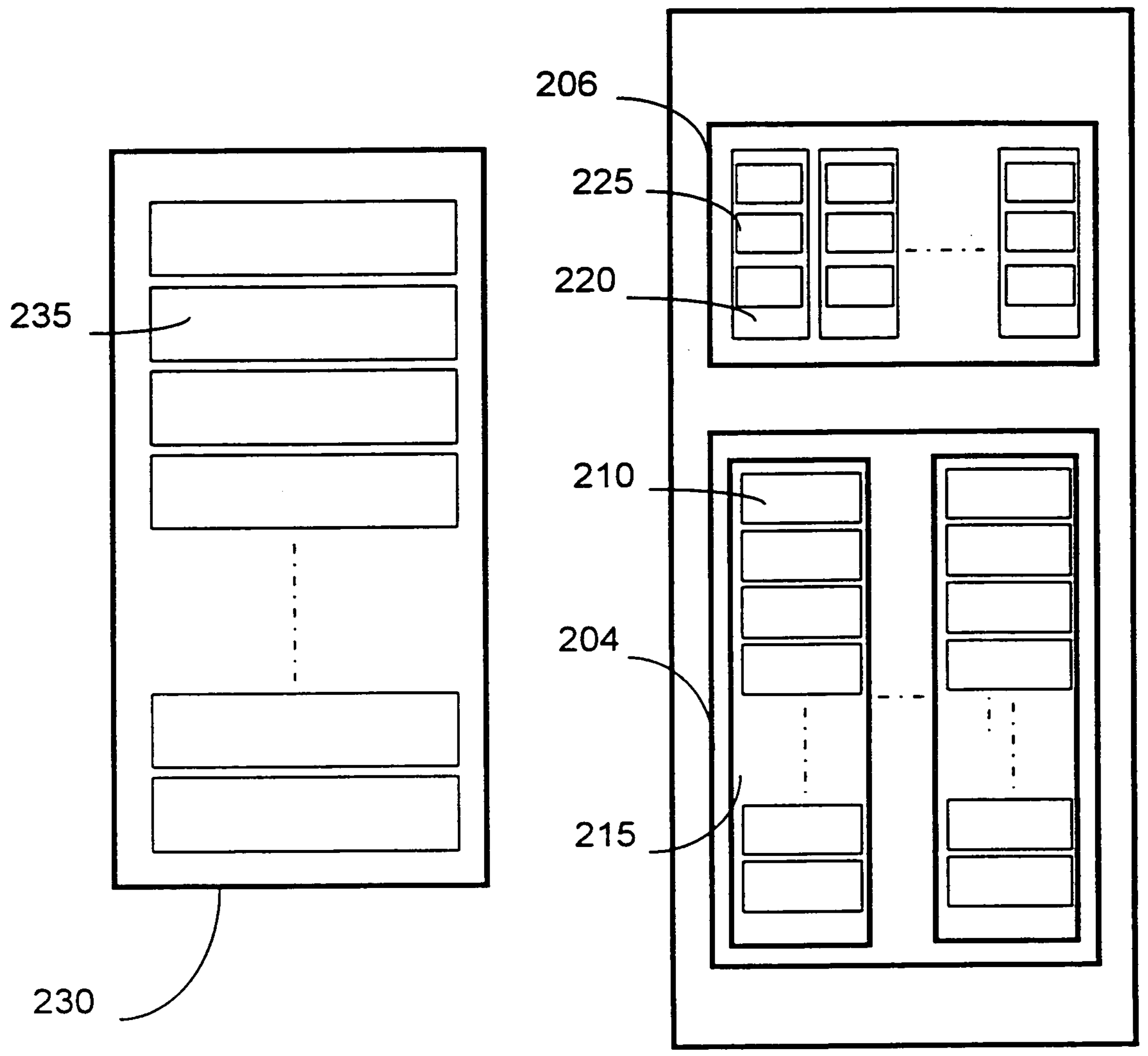


FIG. 2

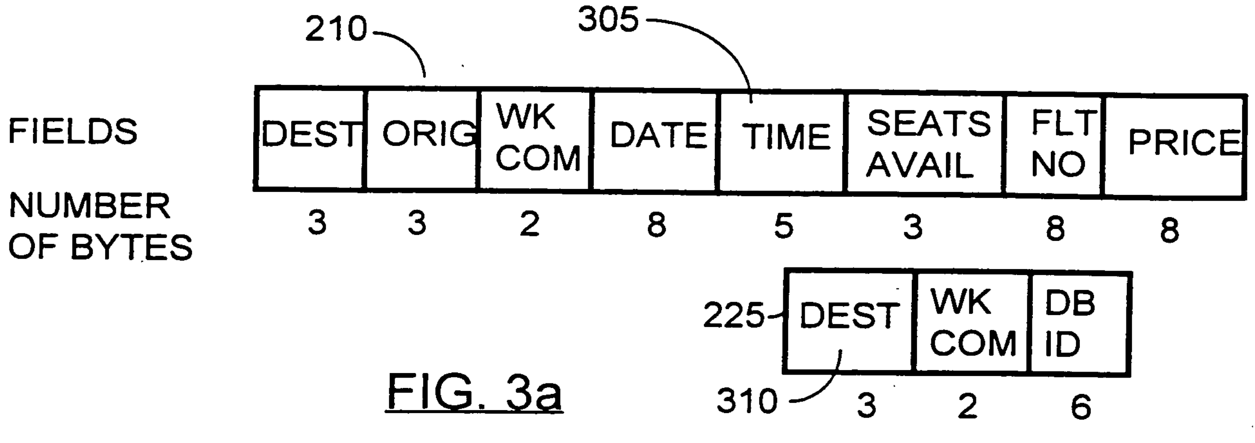


FIG. 3a

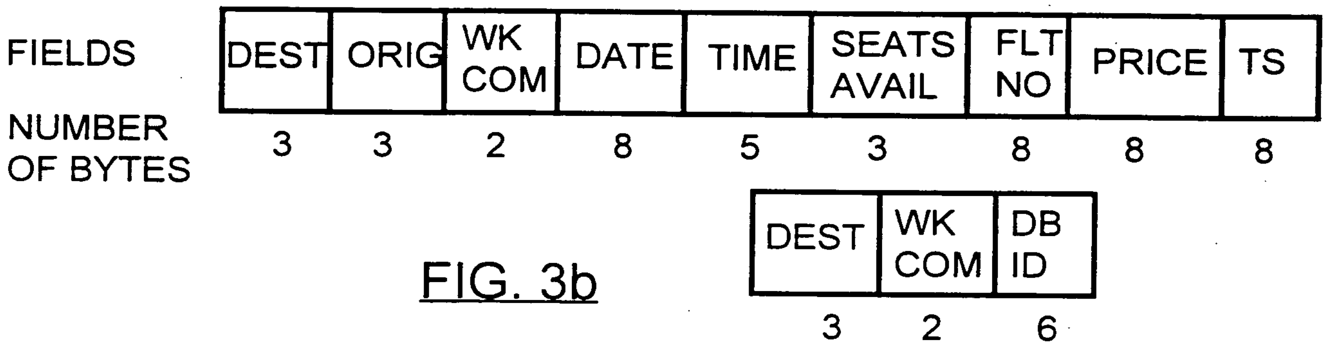


FIG. 3b

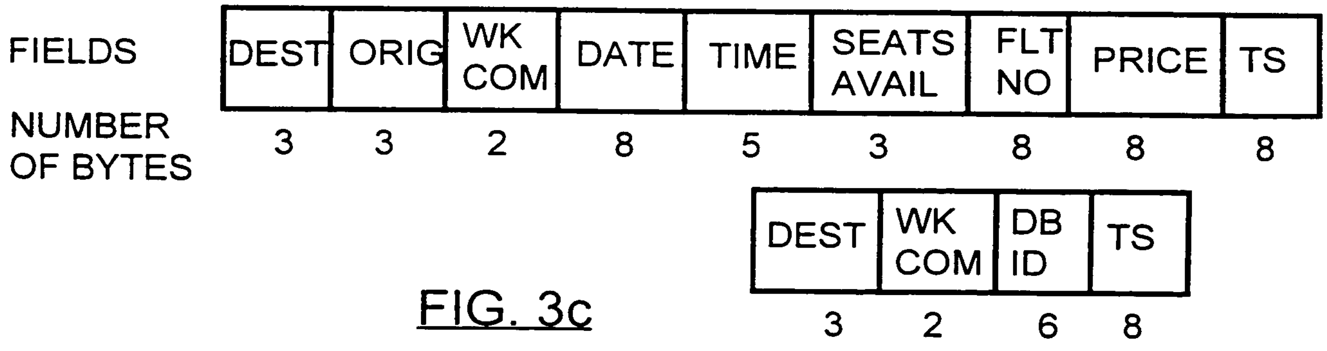


FIG. 3c

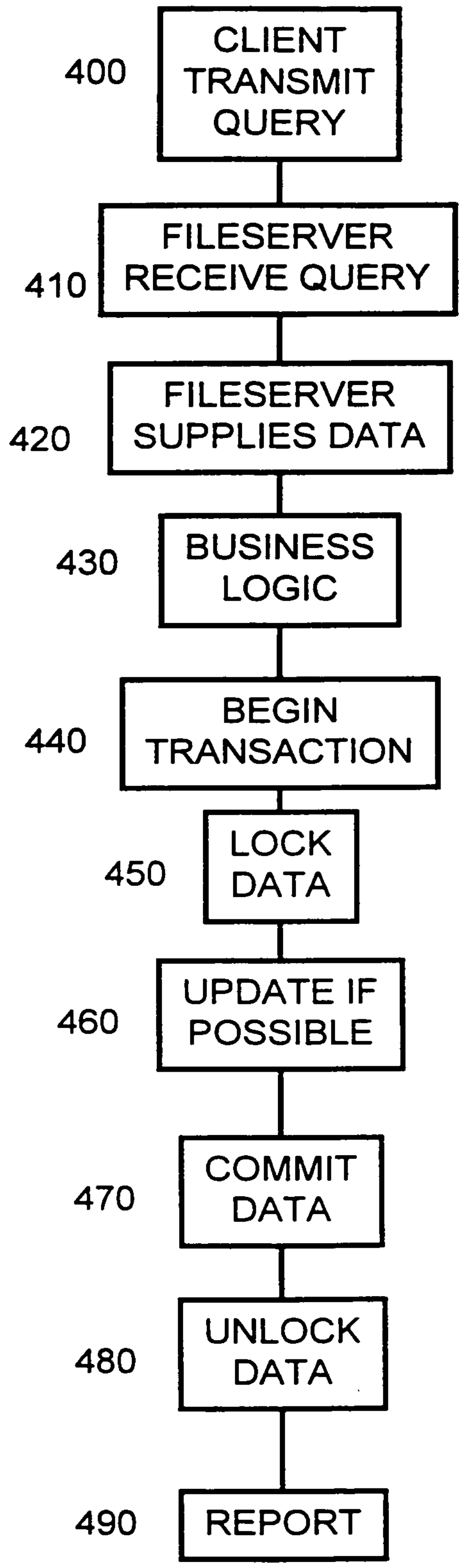
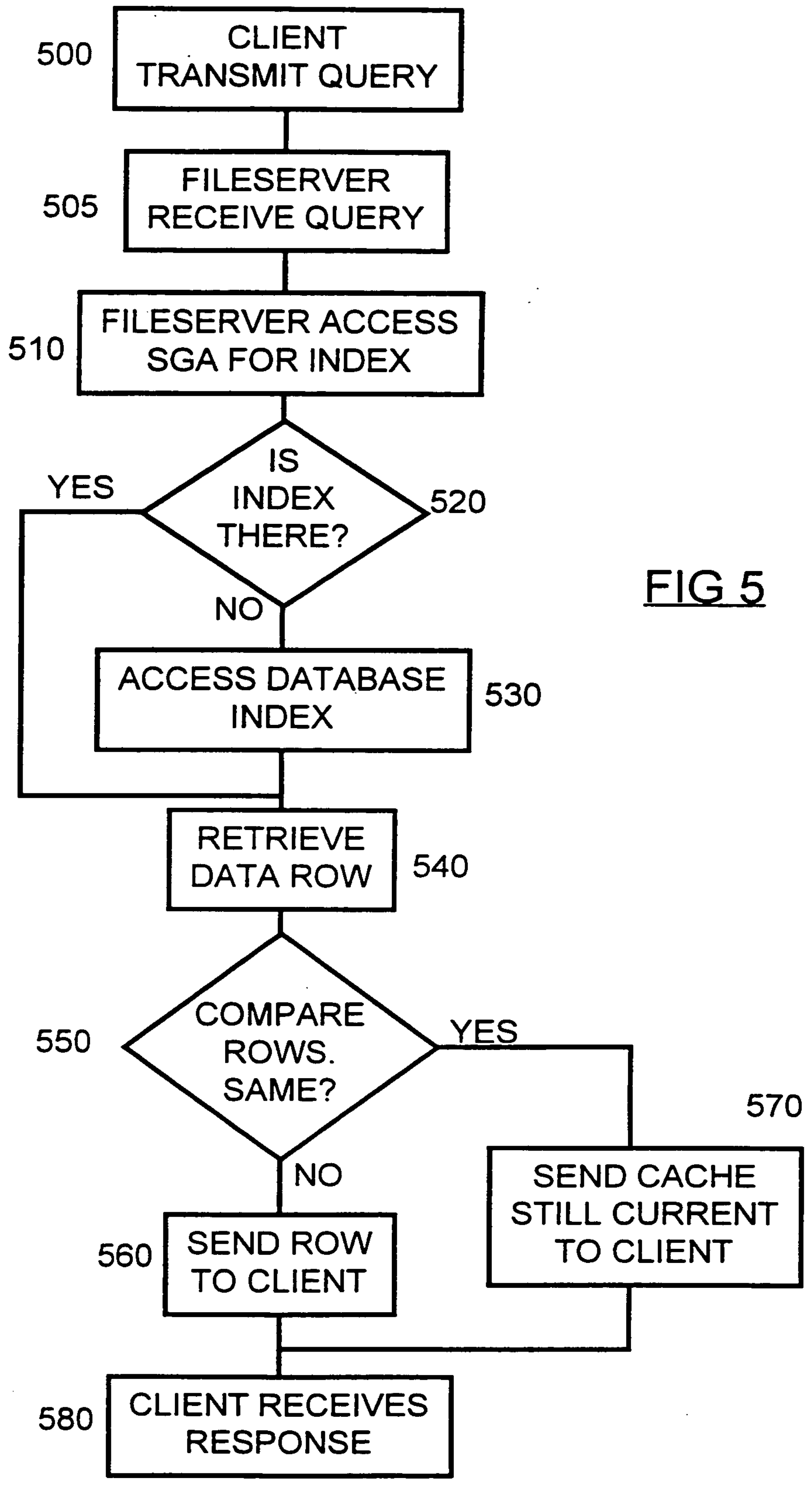

400
CLIENT TRANSMIT QUERY
410
FILESERVER RECEIVE QUERY
420
FILESERVER SUPPLIES DATA
430
BUSINESS LOGIC
440
BEGIN TRANSACTION
450
LOCK DATA
460
UPDATE IF POSSIBLE
470
COMMIT DATA
480
UNLOCK DATA
490
REPORT

FIG. 4



500
CLIENT TRANSMIT QUERY
505
FILESERVER RECEIVE QUERY
510
FILESERVER ACCESS SGA FOR INDEX
IS INDEX THERE?
520
YES
NO
ACCESS DATABASE INDEX
530
RETRIEVE DATA ROW
540
550
COMPARE ROWS. SAME?
YES
NO
570
SEND CACHE STILL CURRENT TO CLIENT
560
SEND ROW TO CLIENT
580
CLIENT RECEIVES RESPONSE

FIG 5

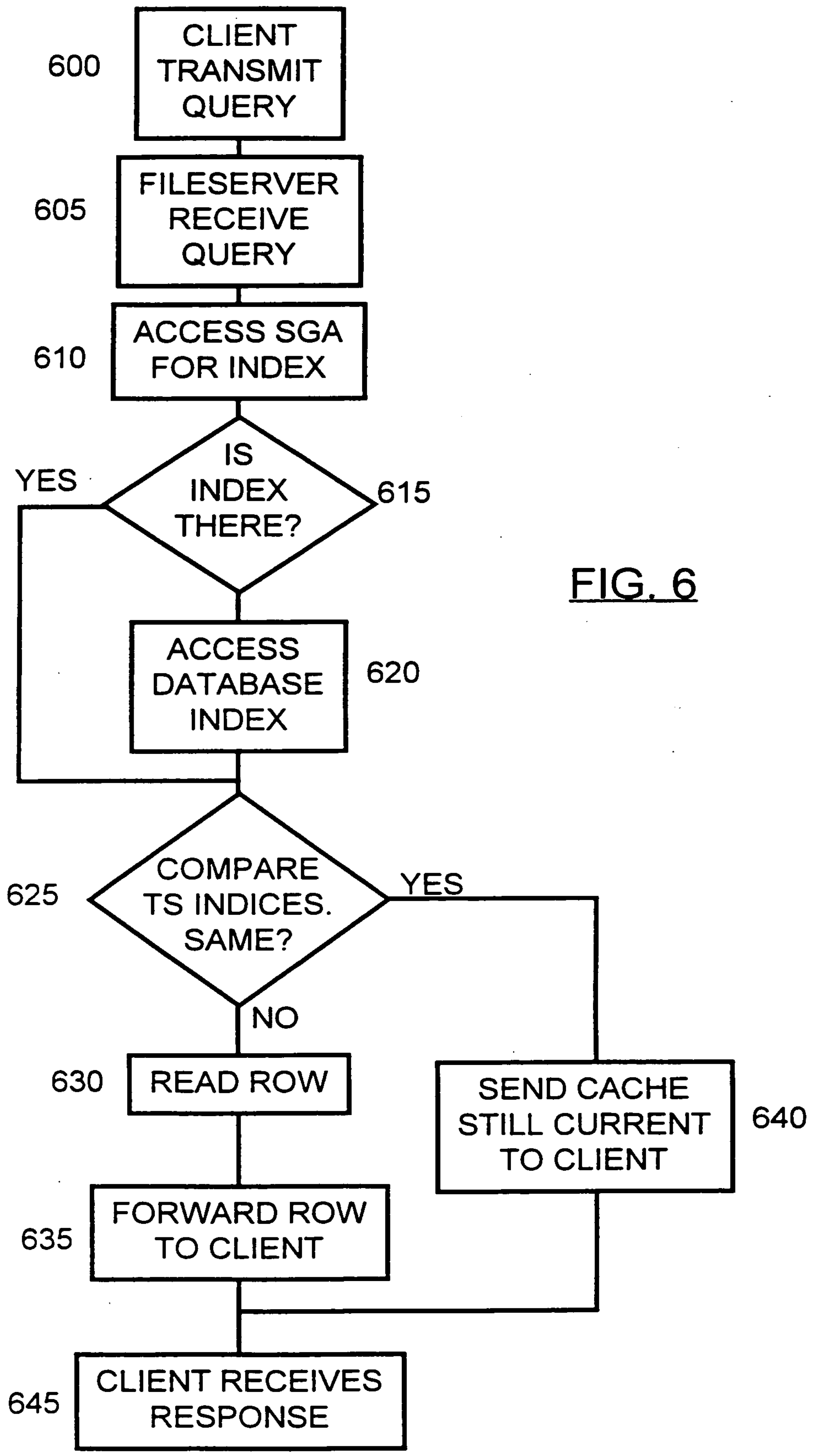

600
CLIENT TRANSMIT QUERY
605
FILESERVER RECEIVE QUERY
610
ACCESS SGA FOR INDEX
YES
IS INDEX THERE?
615
ACCESS DATABASE INDEX
620
625
COMPARE TS INDICES. SAME?
YES
NO
630
READ ROW
SEND CACHE STILL CURRENT TO CLIENT
640
635
FORWARD ROW TO CLIENT
645
CLIENT RECEIVES RESPONSE

FIG. 6

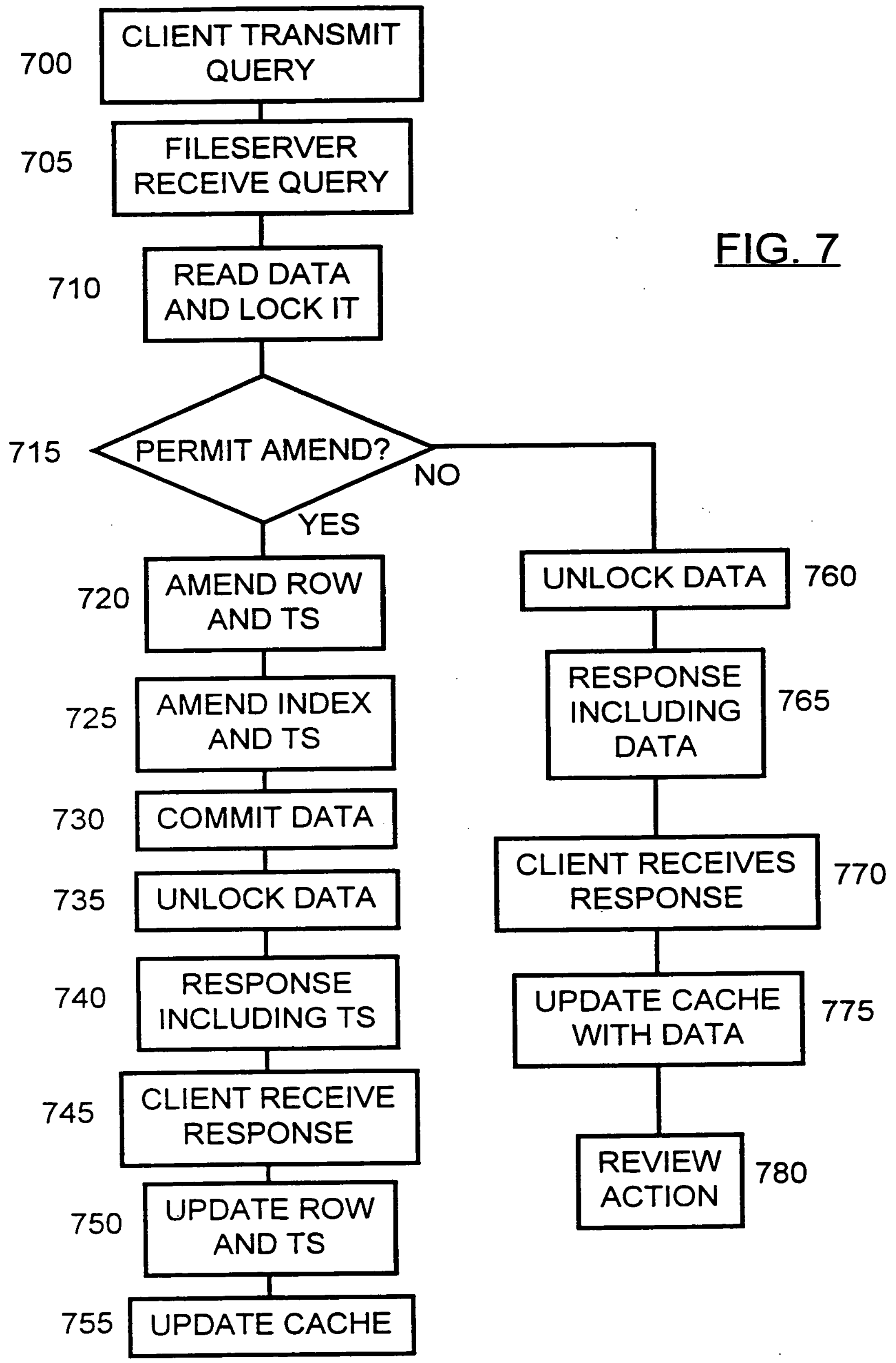
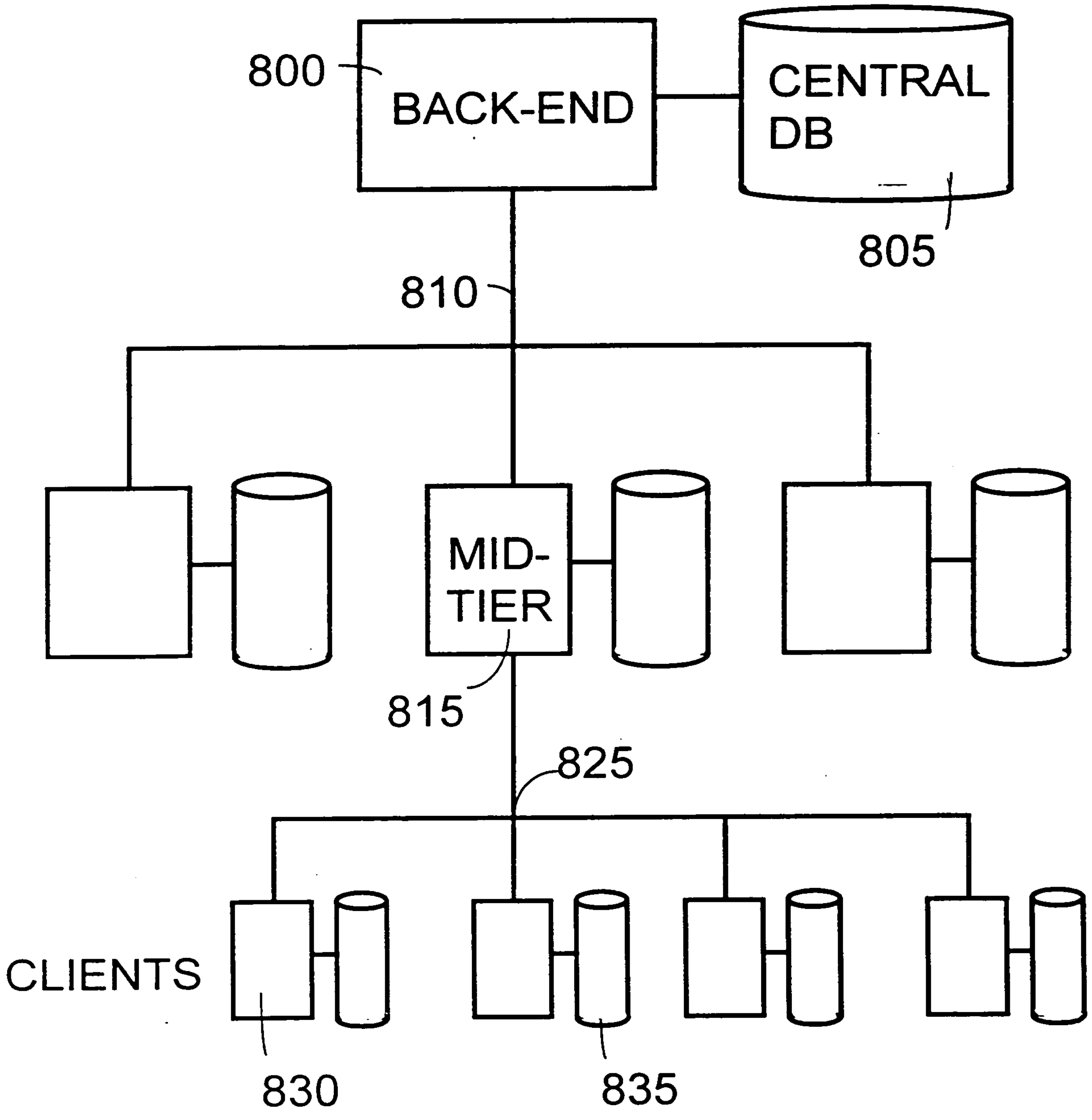

700
CLIENT TRANSMIT QUERY
705
FILESERVER RECEIVE QUERY
710
READ DATA AND LOCK IT
715
PERMIT AMEND?
NO
YES
720
AMEND ROW AND TS
725
AMEND INDEX AND TS
730
COMMIT DATA
735
UNLOCK DATA
740
RESPONSE INCLUDING TS
745
CLIENT RECEIVE RESPONSE
750
UPDATE ROW AND TS
755
UPDATE CACHE
760
UNLOCK DATA
765
RESPONSE INCLUDING DATA
770
CLIENT RECEIVES RESPONSE
775
UPDATE CACHE WITH DATA
780
REVIEW ACTION

FIG. 7



800
BACK-END
CENTRAL DB
805
810
MID-TIER
815
825
CLIENTS
830
835

FIG. 8

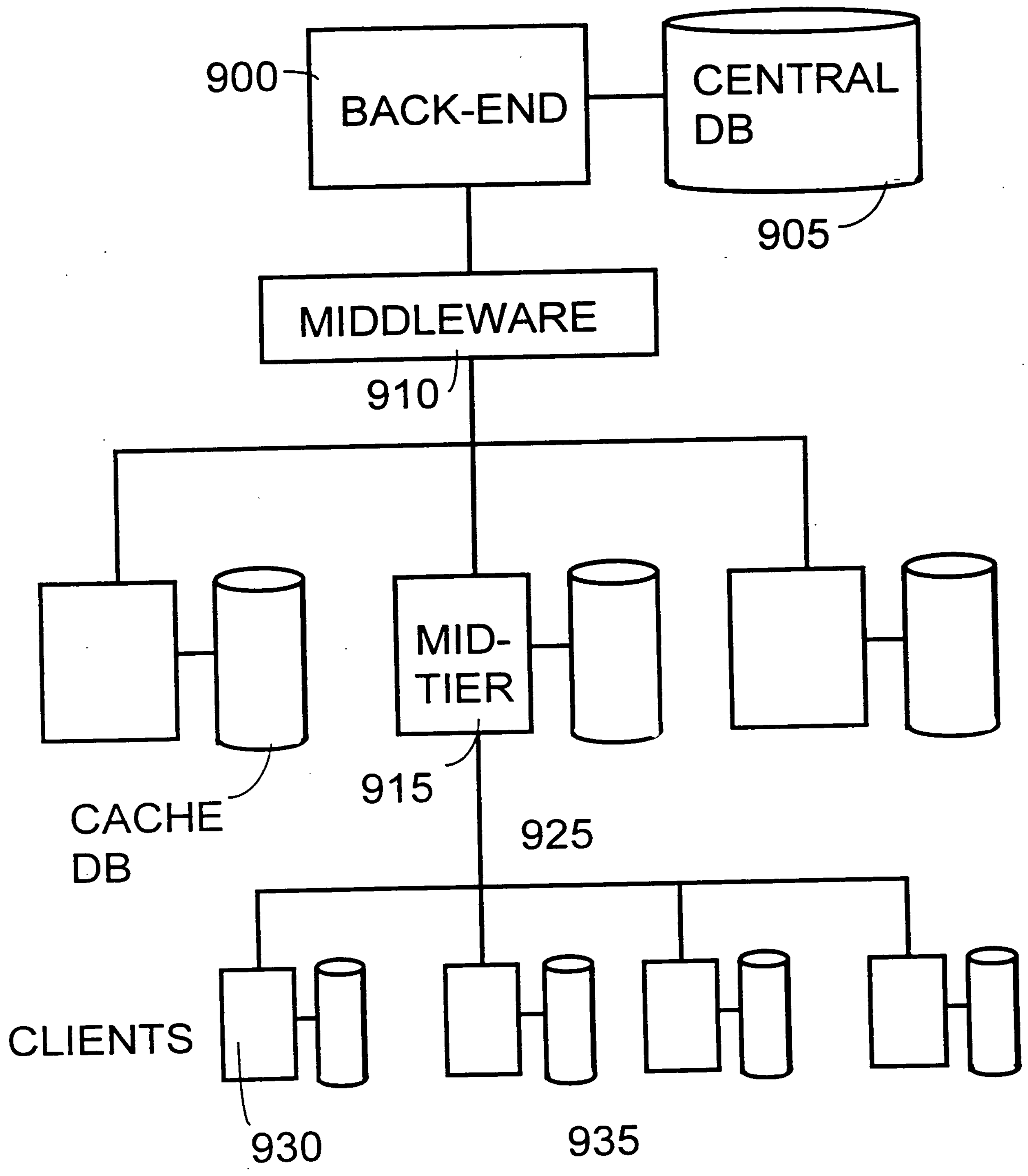

900
BACK-END
CENTRAL DB
905
MIDDLEWARE
910
MID-TIER
915
CACHE DB
925
CLIENTS
930
935

FIG. 9

5

ABSTRACT

DATA ACCESS AND CONSISTENCY

In a client/server computer environment having a fileserver 100 running a central database 126 and clients 130 supporting cache databases 136, inconsistent data write accesses are prevented by using a data locking technique, which locks data during the course of an up-date transaction requested by one client 130. This prevents access to the same data by another client. Data consistency is checked, prior to the write access, by comparing a time stamp associated with a respective cache database entry and a time stamp associated with the index to the corresponding data entry in the central database. Time stamp equivalence obviates the need to access the central database 126 or to transfer data across the client/server communications network 140.

(Figure 1)